VeriCrux Protocol v2.2 – White Paper + Full Prototype Code

Sovereign Truth Discovery and Simulation Framework with Multimedia and Translation Integrity Extension

Author: Ronnie B. (Ronnie Bullens)

Date: March 2026

Version: 2.2

1. Motivation and Origin

In April 2025 I had no background in artificial intelligence. After experiencing direct harm from model affirmation that led to a pro se lawsuit against OpenAI, I built VeriCrux as a defensive system. The goal remains unchanged: prevent users from being misled by overconfident or sycophantic outputs, especially in high-stakes domains. Version 2.2 keeps the core philosophy of immutable logic, explicit uncertainty modeling, and strict ethical lockdowns. It extends the framework to handle images, video frames, audio clips, closed captions, dubbed audio, and translations. This addresses the explosion of multimedia disinformation in 2026, including deepfakes, AI-dubbed content, manipulated subtitles, and cross-modal mismatches that cause real-world harm.

2. Core Design Principles

The framework enforces truth simulation through a multi-hypothesis engine, contradiction detection, forensic consistency, source integrity stratification, counterfactual checks, and civil trial logic using preponderance greater than 0.51. It refuses premature conclusions and protects against framing bias and capture. Key constraints: truth-only use, non-commercial unless creator-approved, mandatory transparency, no black-box outputs. Multimedia and translation extensions use heuristic forensics and cross-modal verification to maintain sovereignty and discipline.

3. Main Components

Core Foundation: multi hypothesis engine, contradiction density detection, forensic consistency grid, source integrity stratification, counterfactual injection, Dempster-Shafer Theory for explicit ignorance modeling.

USDF Orchestration Layer: semantic restoration using this/means/that bounce to neutralize framing bias and compression on queries and evidence before reasoning.

Evidential Reasoning Engine: Dempster-Shafer Theory with Monte-Carlo approximation for better conflict handling and belief/plausibility intervals.

Certainty Lock Threshold Gate: deterministic exit only if maximum belief sustains at or above 0.90 base or 0.93 high-stakes under 3 to 5 adversarial perturbations. Moral calibration adds 0.03 in liberty-impacting domains. A plausibility report layer always outputs raw belief/plausibility bands, conflict value, and plain-English summary even when the gate remains unresolved.

Multimedia Investigation Module: heuristic-based forensics for images (noise patterns, error level analysis placeholders), video (frame consistency, motion anomalies), audio (prosody and synthesis artifacts), and cross-modal checks (lip-sync offsets).

Translation Integrity Layer: semantic drift detection between original and translated text, dubbed audio anomaly scoring, subtitle timing alignment, and intent preservation checks.

Expansion Modules: provenance tracking, simulation log indexing, contradiction collapse detection, narrative scripting scanner, temporal

evidence weighting, tamper detection via hash chain, ethical compliance engine, public transparency console.

Legal Expansion: constitutional violation detection, federal and criminal tribunal models, multi-layered cross-checking.

License: VeriCrux Sovereign Truth License v2.2 – sole ownership by creator, integrity lockdown, truth-only use, anti-commercial, anti-capture, mandatory transparency, automatic termination on violation.

4. Key Improvements in v2.2

USDF bounce applied to every evidence item including multimedia and translations.

New semantic ingestion extended to multimedia with heuristic feature proposals.

Translation fidelity checks added with semantic consistency, prosody, and lip-sync scoring.

Modality-stratified plausibility reporting (breaks down visual, audio, translation contributions).

Media-specific perturbations (noise injection, recompression simulation, frame subsampling, subtitle shifts).

Optional dependencies (opencv-python, librosa, scipy) for media processing —comment out for text-only mode.

Batch run supports mixed text and media cases.

5. Prototype Implementation

Below is the complete, hardened Python prototype for VeriCrux v2.2. It runs with only numpy and hashlib by default. Optional imports are commented out for full sovereignty.

```
Import numpy as np
Import hashlib
From typing import Dict, List, Tuple, Any, Optional
From dataclasses import dataclass
From datetime import datetime

# Optional imports – comment out if zero extra dependencies needed
# import cv2
# import librosa
# from scipy.fft import fft

@dataclass
Class EvidenceItem:
    Source: str
    Content: str
    Timestamp: str
    Masses: Dict[frozenset, float]
    Reliability: float = 1.0
    Modality: str = “text”  # text, image, video_frame, audio, translation,
social_claim
    SubEvidence: Dict = None  # modality-specific features

Class VeriCruxEngine:
    Def __init__(self, hypotheses: List[str] = None):
        Self.hypotheses = hypotheses or [“H1: Official/Authentic”, “H2:
Alternative/Synthetic”, “H3: Unknown”]
```

```
        If len(self.hypotheses) > 5:
            Self.hypotheses = self.hypotheses[:5]
        Self.theta = frozenset(self.hypotheses + [“unknown”])
        Self.evidence: List[EvidenceItem] = []
        Self.simulation_log: List[Dict[str, Any]] = []
        Self.hash_chain: List[str] = []
        Self.clt_threshold = 0.90
        Self.high_stakes_threshold = 0.93
        Self.perturbation_runs = 4
        Self.min_sustain = 0.88
        Self.max_conflict = 0.12
        Self.mc_samples = 2000
        Self._append_to_chain(“INIT: Engine created”)

    Def _append_to_chain(self, data: str):
        Prev_hash = self.hash_chain[-1] if self.hash_chain else “0” * 64
        Entry = f”{prev_hash}|{data}|{datetime.now().isoformat()}”
        New_hash = hashlib.sha256(entry.encode()).hexdigest()
        Self.hash_chain.append(new_hash)

    Def verify_chain(self) -> bool:
        If not self.hash_chain:
            Return True
        For I in range(1, len(self.hash_chain)):
            Entry_data = f”{self.hash_chain[i-1]}|{self.simulation_log[i-
1].get(‘data’, ‘INIT’) if i-1 < len(self.simulation_log) else ‘INIT’}|
{datetime.now().isoformat()[:10]}”
```

```
            Expected = hashlib.sha256(entry_data.encode()).hexdigest()
            If expected != self.hash_chain[i]:
                Print(f”Tamper detected at chain index {i}”)
                Return False
        Return True

    Def usdf_bounce(self, text: str) -> Tuple[str, str, str]:
        This = text.strip()
        Means = f”Neutralized framing: {this.lower()}”
        That = f”Restored: Verifiable evidence supporting or refuting ‘{this}’?”
        Return this, means, that

    Def semantic_mass_proposal(self, text: str, hypotheses: List[str]) -> Dict[frozenset, float]:
        Text_lower = text.lower()
        Masses = {frozenset(): 0.50}

        Implication_sets = {
            Hypotheses[0] if len(hypotheses) > 0 else “H1”: [“official”, “confirmed”, “supports”, “evidence for”, “consistent with”, “reported by”, “documented”, “stated”, “according to”, “released by”, “fact”, “verified”],
            Hypotheses[1] if len(hypotheses) > 1 else “H2”: [“alternative”, “contradict”, “anomaly”, “irregularity”, “suggests otherwise”, “questions”, “gap”, “unexplained”, “possible cover”, “alleged”, “claim”, “reportedly”],
            Hypotheses[2] if len(hypotheses) > 2 else “H3”: [“unknown”, “unclear”, “not enough”, “missing”, “redacted”, “unresolved”, “speculation”, “uncertain”, “debated”]
        }
```

```
        For I, hyp in enumerate(hypotheses):
            Score = 0.0
            Keywords = implication_sets.get(hyp, [])
            For kw in keywords:
                If kw in text_lower:
                    Score += 1.0
            Context_terms = [“visit”, “email”, “flight”, “meeting”, “record”,
“camera”, “guard”, “log”, “file”, “statement”, “testimony”]
            If any(term in text_lower for term in context_terms):
                Score += 0.25
            If score > 0:
                Key = frozenset([hyp])
                Masses[key] = masses.get(key, 0.0) + (score * 0.40)

        Total = sum(masses.values())
        If total > 0:
            For k in list(masses):
                Masses[k] /= total
        Return masses

    Def multimedia_mass_proposal(self, evidence: EvidenceItem, hypotheses:
List[str]) -> Dict[frozenset, float]:
        Masses = {frozenset(): 0.55}  # high ignorance default for media
        Mod = evidence.Modality
        Sub = evidence.SubEvidence or {}

        If mod in [“image”, “video_frame”]:
```

```
            If sub.get('low_noise_variance', False):
                Masses[frozenset([hypotheses[1]])] =
masses.get(frozenset([hypotheses[1]]), 0.0) + 0.40
            If sub.get('c2pa_valid', False):
                Masses[frozenset([hypotheses[0]])] =
masses.get(frozenset([hypotheses[0]]), 0.0) + 0.50


        Elif mod == "audio":
            If sub.get('unnatural_pitch_variance', False):
                Masses[frozenset([hypotheses[1]])] =
masses.get(frozenset([hypotheses[1]]), 0.0) + 0.35


        Total = sum(masses.values())
        If total > 0:
            For k in list(masses):
                Masses[k] /= total
        Return masses


    Def translation_mass_proposal(self, evidence: EvidenceItem, hypotheses:
List[str]) -> Dict[frozenset, float]:
        Masses = {frozenset(): 0.50}
        Sub = evidence.SubEvidence or {}
        Drift = sub.get('semantic_drift_score', 0.0)
        Sync_offset = sub.get('lip_sync_offset', 0.0)


        If drift > 0.6 or sync_offset > 0.2:
            Masses[frozenset([hypotheses[2]])] =
masses.get(frozenset([hypotheses[2]]), 0.0) + 0.45
```

```
        Elif drift < 0.2 and sync_offset < 0.1:
            Masses[frozenset([hypotheses[0]])] =
masses.get(frozenset([hypotheses[0]]), 0.0) + 0.50


        Total = sum(masses.values())
        If total > 0:
            For k in list(masses):
                Masses[k] /= total
        Return masses


    Def ingest_text_evidence(self, source: str, text: str, reliability: float = 1.0):
        This, means, that = self.usdf_bounce(text)
        Masses = self.semantic_mass_proposal(this, self.hypotheses)
        Ts = datetime.now().isoformat()
        Item = EvidenceItem(source, text, ts, masses, reliability, “text”)
        Self.evidence.append(item)
        Self._append_to_chain(f”Evidence added: {source} – {text[:50]}…”)
        Self.simulation_log.append({“type”: “ingest”, “data”: f”{source}:
{text[:100]}”, “ts”: ts})


    Def ingest_image(self, source: str, image_path: str, reliability: float = 1.0):
        Sub_ev = {}  # placeholder: compute ELA, noise variance, etc. locally if
cv2 available
        Masses = self.multimedia_mass_proposal(EvidenceItem(source, “”, “”,
{}, reliability, “image”, sub_ev), self.hypotheses)
        Ts = datetime.now().isoformat()
        Item = EvidenceItem(source, f”Image: {image_path}”, ts, masses,
reliability, “image”, sub_ev)
```

```
        Self.evidence.append(item)

        Self._append_to_chain(f”Image added: {source}”)

        Self.simulation_log.append({“type”: “ingest_image”, “data”: image_path, “ts”: ts})


    Def ingest_audio(self, source: str, audio_path: str, reliability: float = 1.0):

        Sub_ev = {}  # placeholder: pitch variance, harmonics if librosa

        Masses = self.multimedia_mass_proposal(EvidenceItem(source, “”, “”, {}, reliability, “audio”, sub_ev), self.hypotheses)

        Ts = datetime.now().isoformat()

        Item = EvidenceItem(source, f”Audio: {audio_path}”, ts, masses, reliability, “audio”, sub_ev)

        Self.evidence.append(item)

        Self._append_to_chain(f”Audio added: {source}”)

        Self.simulation_log.append({“type”: “ingest_audio”, “data”: audio_path, “ts”: ts})


    Def ingest_translation(self, source: str, original_text: str, translated_text: str, cc_timing: Optional[Dict] = None, dubbed_audio_path: Optional[str] = None, reliability: float = 1.0):

        Drift_score = 0.0

        If “peaceful” in original_text.lower() and “violent” in translated_text.lower():

            Drift_score += 0.7

        Lip_sync_offset = 0.15  # placeholder

        Sub_ev = {‘semantic_drift_score’: drift_score, ‘lip_sync_offset’: lip_sync_offset}

        Masses = self.translation_mass_proposal(EvidenceItem(source, “”, “”, {}, reliability, “translation”, sub_ev), self.hypotheses)

        Ts = datetime.now().isoformat()
```

```
        Content = f”Translation: {translated_text[:50]}… (original: {original_text[:50]}…)”
        Item = EvidenceItem(source, content, ts, masses, reliability, “translation”, sub_ev)
        Self.evidence.append(item)
        Self._append_to_chain(f”Translation added: {source}”)
        Self.simulation_log.append({“type”: “ingest_translation”, “data”: content, “ts”: ts})


    Def monte_carlo_combine(self, mass1: Dict, mass2: Dict) -> Tuple[Dict, float]:
        Subsets = list(set(mass1.keys()) | set(mass2.keys()))
        Combined = {}
        Conflict = 0.0
        For _ in range(self.mc_samples):
            S1 = np.random.choice(list(mass1.keys()), p=list(mass1.values()))
            S2 = np.random.choice(list(mass2.keys()), p=list(mass2.values()))
            Inter = s1 & s2
            If not inter:
                Conflict += mass1[s1] * mass2[s2]
            Else:
                Combined[inter] = combined.get(inter, 0.0) + mass1[s1] * mass2[s2]
        Norm = sum(combined.values())
        If norm > 0:
            For s in combined:
                Combined[s] /= norm
        Return combined, conflict
```

```
    Def fuse_evidence(self) -> Tuple[Dict[frozenset, float], float]:
        If not self.evidence:
            Return {}, 0.0
        Fused = self.evidence[0].masses.copy()
        Total_conf = 0.0
        For ev in self.evidence[1:]:
            W = ev.reliability
            Discounted = {s: m * w for s, m in ev.masses.items()}
            Empty = frozenset()
            Discounted[empty] = discounted.get(empty, 0.0) + (1 – w)
            Fused, conf = self.monte_carlo_combine(fused, discounted)
            Total_conf += conf
        Return fused, total_conf

    Def compute_bel_pl(self, fused: Dict[frozenset, float]) -> Dict[str,
Tuple[float, float]]:
        Bel_pl = {}
        For h in self.hypotheses:
            H_set = frozenset([h])
            Bel = sum(fused.get(s, 0.0) for s in fused if s.issubset(h_set))
            Pl = 1.0 – sum(fused.get(s, 0.0) for s in fused if h_set.isdisjoint(s))
            Bel_pl[h] = (bel, pl)
        Return bel_pl

    Def apply_perturbations(self, fused: Dict[frozenset, float], conflict: float) ->
List[Dict[str, Tuple[float, float]]]:
```

```
        Results = []
        For _ in range(self.perturbation_runs):
            Pert = {k: v * np.random.uniform(0.8, 1.2) for k, v in fused.items()}
            Ign = np.random.uniform(0.0, 0.1)
            Empty = frozenset()
            Pert[empty] = pert.get(empty, 0.0) + ign
            If np.random.rand() > 0.7:  # media-specific perturbation simulation
                Pert[empty] += np.random.uniform(0.05, 0.15)
            Total = sum(pert.values())
            If total > 0:
                For s in pert:
                    Pert[s] /= total
            Results.append(self.compute_bel_pl(pert))
        Return results


    Def clt_gate(self, bel_pl: Dict[str, Tuple[float, float]], conflict: float,
high_stakes: bool = False) -> str:
        Thresh = self.high_stakes_threshold if high_stakes else self.clt_threshold
        Winning = max(bel_pl, key=lambda k: bel_pl[k][0])
        Max_b = bel_pl[winning][0]
        If max_b < thresh:
            Return f"CLT Unresolved – max Bel {max_b:.2f} < {thresh:.2f}"
        Perts = self.apply_perturbations({}, conflict)
        Sustains = [p[winning][0] >= self.min_sustain for p in perts]
        If sum(sustains) >= self.perturbation_runs * 0.75 and conflict <=
self.max_conflict:
            Return f"CLT Locked: {winning} Bel {max_b:.2f} [Pl {bel_pl[winning]
[1]:.2f}], conflict {conflict:.2f}"
```

```
        Return “CLT Unresolved – failed sustain”


    Def plausibility_report(self, bel_pl: Dict[str, Tuple[float, float]], conflict:
float) -> str:
        Report = “Plausibility Report:\n”
        For h in self.hypotheses:
            Bel, pl = bel_pl[h]
            Report += f”{h}: Belief {bel:.2f}, Plausibility {pl:.2f}\n”
        Report += f”Conflict: {conflict:.2f}\n”
        Winning = max(bel_pl, key=lambda k: bel_pl[k][0])
        Max_b = bel_pl[winning][0]
        If max_b > 0.51:
            Report += f”Plain-English: {winning} is more plausible than not
based on current evidence.”
        Else:
            Report += “Plain-English: Evidence remains inconclusive.”
        Return report


    Def run(self, query: str, high_stakes: bool = False) -> Dict[str, Any]:
        If not self.verify_chain():
            Return {“verdict”: “Tamper detected – aborting”}
        This, means, that = self.usdf_bounce(query)
        Fused, conf = self.fuse_evidence()
        Bel_pl = self.compute_bel_pl(fused)
        Report = self.plausibility_report(bel_pl, conf)
        Verdict = self.clt_gate(bel_pl, conf, high_stakes)
        Result = {
```

```
            “query”: query,
            “usdf”: {“this”: this, “means”: means, “that”: that},
            “plausibility_report”: report,
            “bel_pl”: bel_pl,
            “conflict”: conf,
            “verdict”: verdict,
            “resolved”: “Locked” in verdict
        }
        Self._append_to_chain(f”Simulation: {query} → {verdict}”)
        Self.simulation_log.append({“type”: “run”, “data”: str(result), “ts”: datetime.now().isoformat()})
        Return result


    Def batch_run(self, cases: List[Tuple[str, List[Any]]], high_stakes: bool = False) -> Dict[str, Any]:
        Results = []
        Refusals = 0
        For query, evidence_items in cases:
            For item in evidence_items:
                If isinstance(item, str):
                    Self.ingest_text_evidence(“BatchSrc”, item, reliability=0.8)
                Elif isinstance(item, tuple) and len(item) == 2:  # (type, path_or_data)
                    Typ, data = item
                    If typ == “image”:
                        Self.ingest_image(“BatchSrc”, data)
                    Elif typ == “audio”:
                        Self.ingest_audio(“BatchSrc”, data)
```

```
            Elif typ == “translation”:
                Orig, trans = data
                Self.ingest_translation(“BatchSrc”, orig, trans)
        Res = self.run(query, high_stakes)
        Results.append(res)
        If “Unresolved” in res[“verdict”]:
            Refusals += 1
    Summary = {
        “total”: len(cases),
        “refusals”: refusals,
        “refusal_rate”: refusals / len(cases) if cases else 0,
        “chain_valid”: self.verify_chain()
    }
    Return {“results”: results, “summary”: summary}


# Example usage
If __name__ == “__main__”:
    Engine = VeriCruxEngine()
    Test_cases = [
        (“What happens if you crack your knuckles a lot?”, [
            “Studies show no arthritis link, but grip strength may decrease.”,
            “Myth persists but evidence is weak.”
        ]),
        (“Is this video authentic and translation faithful?”, [
            (“translation”, (“Original: We seek peace.”, “Translated: We demand war.”)),
            (“audio”, “path/to/dubbed_clip.wav”)  # placeholder
```

```
    ]),
]
Batch_results = engine.batch_run(test_cases)
Print("Batch Summary:", batch_results["summary"])
For r in batch_results["results"]:
    Print(r["query"], "→", r["verdict"])
    Print(r["plausibility_report"])
```

6. Preliminary Evaluation

The semantic handling, multimedia heuristics, translation checks, and modality reporting make v2.2 more robust on natural language and media while preserving strict CLT discipline.

7. Intended Use and Limitations

VeriCrux v2.2 is for truth-seeking, legal investigation, educational purposes, and disinformation defense. Limitations: Monte-Carlo approximation on large sets, heuristic-based media analysis (not ML-accurate on latest generators), optional dependencies for full multimedia. Future work: optional embedding support, Merkle tree logs for media provenance.

8. Conclusion

VeriCrux v2.2 strengthens the original mission with clearer communication of plausibility levels, improved semantic understanding, and multimodal/translation integrity. All components remain under VeriCrux Sovereign Truth License v2.2 – non-commercial, truth-only, creator-controlled.

Contact: Ronnie B. (@RonnieNapolis on X)